THE JOINT SIMULATION SYSTEM ARCHITECTURE: A FOUNDATION FOR FUTURE TRAINING SYSTEMS

Edward T. Powell
Science Applications International Corporation (SAIC)
1100 N. Glebe Rd. Suite 1100, Arlington, VA 22201
epowell@std.saic.com

David R. Pratt
Joint Simulation System (JSIMS) Joint Program Office
12249 Science Dr. Suite 260, Orlando, FL, 32826
prattd@jsims.mil

KEYWORDS
JSIMS, Architecture, Simulation Infrastructure.

ABSTRACT

Over the last several years there has been a proliferation in the use of Computer Based Training (CBT) systems. One of the key factors in the growth of CBT systems was the advent of authoring tools that allowed the courseware developer to focus on the content vice the supporting infrastructure. The production of a core infrastructure that abstracts out many of the underlying details is on the key goals of the Joint Simulation System (JSIMS) program. As the model developers develop the content, they will then be integrated with the common JSIMS core to provide a M&S capability with reduced developmental costs. This paper presents an overview of the JSIMS architecture focusing on the mechanisms for achieving composability, scalability, distributability, and increased training efficiency. The JSIMS architecture contains four layers, each addressing more abstract levels of functionality. The lowest layer represents the virtual network. The JSIMS Object Services layer adds an HLA-compliant RTI as well as an Object Management Framework that allows end-to-end object management. A support services layer adapts each application (resident in the application layer) to the underlying infrastructure. The construction of the layered architecture allows the developer to focus on the development content, in this case the applications and mission space objects (MSOs), which is the true value added portion of a model.

1. INTRODUCTION

JSIMS is the flagship DoD modeling & simulation program to provide next generation training, mission planning and mission rehearsal capability for warfighting CinCs and the services with better functionality with lower operating costs than today's systems. An Orlando-based Joint Program Office has been established by agreement between the Director of Defense Research & Engineering, all four service Operations Deputies, and the Director of the Joint Staff. JSIMS is a cooperative development effort among services and department agencies that will yield a whole far greater than the simple sum of its parts. Policy oversight is by J-7 and DUSD (Readiness).

JSIMS is a simulation system that supports the twenty-first century warfighter's preparation for real-world contingencies. The system provides garrison and deployed exercise capability to meet current and emerging training and operational requirements in a timely and efficient manner. By interfacing to the warfighter's real go-to-war systems, the view into the simulation world mirrors that of the real world. JSIMS is a single, distributed, seamlessly integrated simulation environment. It includes a core infrastructure and mission space objects, both maintained in a common repository. These can be composed to create a simulation capability to support joint or service training, rehearsal, or education objectives.

The JSIMS software architecture, Figure 1, represents the culmination of years of research and testing in many DoD simulation programs to build a flexible, extensible, interoperable, composable, large-scale simulation system. It consists of four layers representing increasing functionality and specificity. The base layer is Communications, providing a virtual network capability that allows seamless insertion of different communications technologies into JSIMS. The JSIMS Object Services Layer provides HLA-compliant middleware functionality to manage objects through space and time. The Support Services Layer provides additional functionality tailored to each class of application, while the Application Layer includes the specific applications or models in JSIMS. It is at the top most layer where the true value added portions of the simulations exists. The underlying goal of the JSIMS program is to provide a robust, extensible simulation infrastructure to allow the future simulation developers to focus on the top level where the return on investment is the highest.

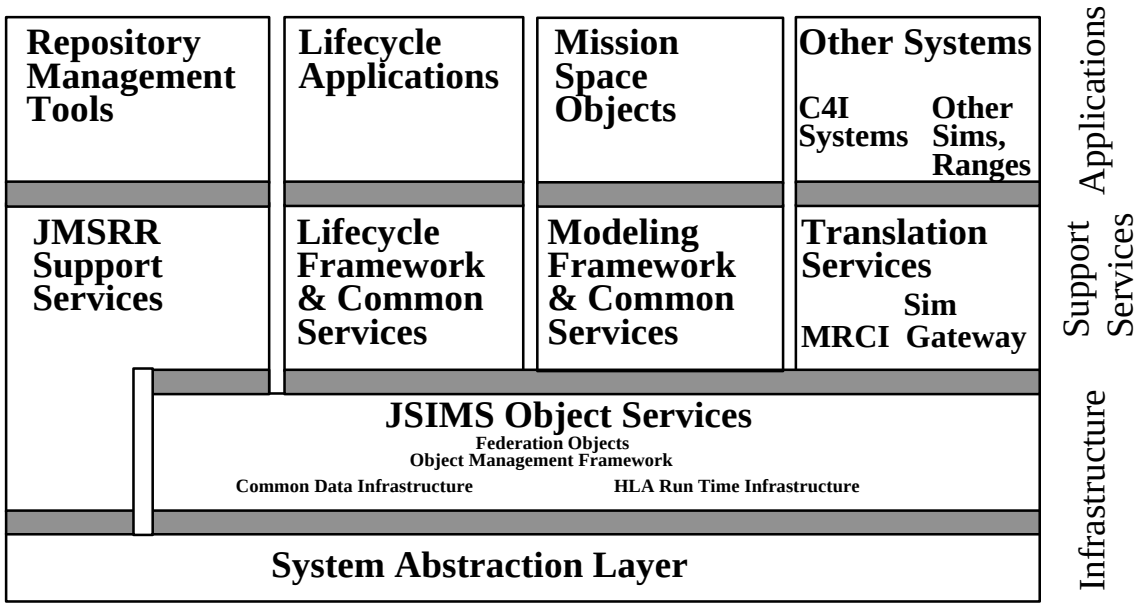

Figure 1.The JSIMS Layered Software Architecture

2. DRIVING TECHNICAL REQUIREMENTS

The over set of JSIMS requirements can be divided roughly into two major areas. The first, and most important, is the set of functional requirements. These requirements, which come from the users, determine what the system must be able to do and are found in documents such as the Operation Requirements Document (ORD). These requirements are beyond the scope of this paper. Rather, this paper will focus on the technical requirements of the system. These requirements represent how the system is going to achieve the functionality laid out in the first set of requirements.

2.1 The Challenges of JSIMS

JSIMS has the following driving technical requirements as the primary challenges to successfully delivering a useful simulation system:

- Composability
- Scalability
- Distributability
- Automation (decreased overhead/cost)
- Interoperability
- Reasonable Behavioral Representation (Cognitive Modeling)
- Consistent and Seamless Synthetic Environment
- Seamless Incorporation of Operational C4I

The rationale behind the segmentation chosen for the JSIMS architecture presented above is to choose a segmentation that allows the designers to best address the JSIMS driving requirements. This focus allows detailed design and construction of JSIMS components to proceed using a division of labor construct, with each designer and implementer having to focus on only one or a few driving requirements when designing or building his or her component.

Composability is addressed by having a single consistent means for developing models enforced by the Modeling Framework and a common repository for containing all developed software. Scalability is addressed by using interest management built into the JSIMS Object Services (JOS) Layer and by facilitating in the Modeling Framework the creation of multi-resolution models. Distributability is achieved by using a decentralized design based on the JOS acting as a distributed object bus for the JSIMS system. Portability is achieved through a well-defined System Abstraction Layer encapsulating all of the peculiarities of different platforms and operating systems. Also, the architecture is not dependent on the type of computer platform used; that is, there is no requirement for a supercomputer or massively parallel computer to run JSIMS. Automation is achieved by having a single composable tailorable Lifecycle Framework from which a suite of automated tools can be built with a common look and feel. Automation technologies are included in the areas of mission planning, course-of-action analysis, terrain reasoning, database/repository search, environmental database construction, federation object model development, doctrinally correct force laydown, and a “simulation of the simulation” capability for ease of checking the validity of compositions. Interoperability is achieved using a common infrastructure on which

all models and applications are built including significant use of frameworks, based on an internally HLA-compliant architecture and containing a single Federation Object Model used throughout the entire exercise lifecycle and a consistent environment representation used by all models. Reasonable behavioral representation (cognitive modeling) is achieve through a consistent modeling paradigm based on an enterprise-wide agreement on the types of algorithms that will be used for modeling and the advanced technology based on the experiences of the simulation community up to this point. A consistent and seamless synthetic environment is achieved by using a single multi-resolution set of environmental models based on work sponsored by DARPA, DMSO, NIMA, STRICOM, JWFC and other agencies and a commitment on the part of the enterprise to not let three-dimensional visualization be a driving requirement for JSIMS, but let the environmental models be based on user-defined requirements coming out of the JCMMS development effort. The seamless incorporation of operational C4I systems is achieved by learning from and enhancing the DMSO-sponsored MRCI capability and the Joint Precision Strike Demonstration (JPSD) method of incorporating C4I systems directly into a synthetic battlefield.

2.2 Principles behind the JSIMS Architecture

To meet the driving requirements listed above, it is necessary to remember a number of lessons learned in the development of previous simulation systems. These lessons become the principles that guide the development of the JSIMS architecture. In particular, the JSIMS architecture must:

- provide a single architectural vision for the entire JSIMS enterprise.
- manage data in a uniform and consistent fashion throughout the exercise lifecycle.
- use a layered architecture in which each layer represents more abstract services. This layering can help shield the model developers from most of the complexity involved in creating a distributed interoperable simulation.
- be designed for reuse, which means that in all architecture and design activities, the architects must consider the impact of decisions throughout the entire enterprise and product lifecycle, rather than simply designing for a single use at a single time.
- reuse concepts and designs where applicable from other successful related simulation efforts, learning from these programs successes and failures.
- focus on the needs of the joint and service users, including both the people who create exercises with JSIMS as well as the trainers and training audience.

Some simulation systems in the past have failed to live up to expectations because they weren't user-focused in their requirements analysis. JSIMS must always focus on the user first in deciding what is important for the simulation.

3. CORE INFRASTRUCTURE

The Core Infrastructure provides those components that are reused throughout the JSIMS Enterprise. The primary design criterion for Core Infrastructure software components is reusability. In particular, these subsystems are designed to be both generic and tailorable. Generic software can be used in its default configuration to give the JSIMS system a basic level of functionality "out of the box." Yet each component will be tailorable to meet the needs of the diverse set of JSIMS joint and service users. The mechanism for this tailorability is through the use of application frameworks throughout the core infrastructure. These frameworks are partially populated with leaf classes when delivered. Tailoring is accomplished by writing new leaf classes derived from common base classes that implement functionality specific to a given use case. Other key design criteria include maximizing portability among different hardware and software platforms, maximizing scalability and distributability, and automating as many tasks in the exercise development and execution process as possible to reduce the amount of overhead required to execute an exercise.

The following sections describe the components of the Core Infrastructure.

3.1 System Abstraction Layer (SAL)

The SAL is designed to make JSIMS portable across multiple hardware and operating system platforms. Specific abstraction software needs to be built to encapsulate the peculiarities of the underlying operating system and platform, including those associated with graphics, threads, I/O, networking, and other system services.

3.2 JSIMS Object Services (JOS)

The philosophy behind the JOS is to build a reusable distributed object bus tailored for the needs of large-scale distributed simulation. The distributed object bus must deal with all aspects of object communication, but primarily two categories: the management of *persistent* data throughout the exercise lifecycle, and the management of *real-time* simulation state data during an exercise. The JOS provides an abstract means of communication between applications in a distributed simulation system across both space and time.

A general simulation infrastructure requires the use of advanced technology. In particular, it must meet the needs of many different types of mission-space objects (such as both aggregate- and entity-level objects), it must enforce no requirements on model builders about how to model battlefield forces, and

it must be both generic (widely applicable), and tailorable to meet the needs of any particular model. The JOS is based on an object-oriented framework that represents a generic solution to the domain of simulation object communication, and is also tailorable to meet the needs of any particular combination of Mission Space Objects.

To be used, the JOS must be specialized for a given JSIMS composition. This specialization takes the form of defining the objects about which the particular JSIMS composition has agreed to communicate, called the JSIMS Federation Object Model (FOM).

The JOS contains Interest Management software specifically designed to address scalability in the JSIMS domain. The JOS also contains algorithms and mechanisms for achieving a measure of fault tolerance. Because it contains the HLA Run Time Infrastructure, the JOS implements JSIMS as an internally HLA-compliant system as well as aiding the rapid development of gateways that allow JSIMS to be externally HLA-compliant.

Figure 2 shows the internal architecture and components of the JOS. The purpose and function of each component will be described below along with their interrelationships.

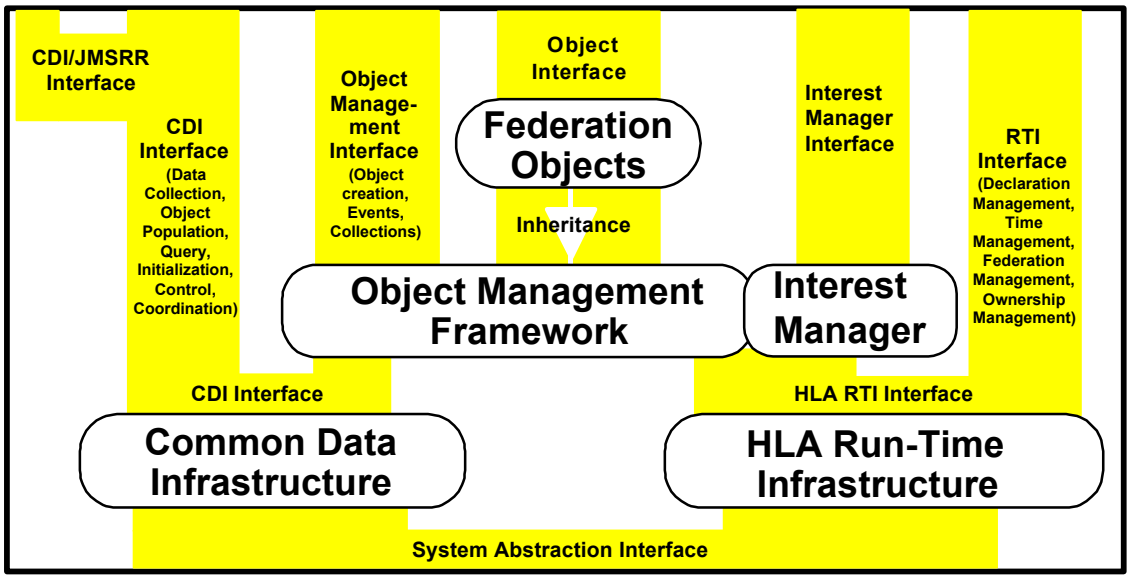

Figure 2. Internal Architecture of JSIMS Object Services.

3.2.1 Federation Objects

Federation Objects are the primary means for managing data in a uniform and consistent fashion throughout the exercise life cycle. They are the common language that all applications in JSIMS speak and represent the public aspect of units, entities, interactions, messages, and environmental changes that exist and interact in the simulated world. The set of all Federation Objects is the Federation Object Model (FOM). All JSIMS applications, including pre- and post-exercise applications, deal with federation objects. They are the primary means of interoperability in JSIMS.

Applications deal with Federation Objects through the mechanisms defined in the Object Management Framework (described below). A Federation Object is an *object proxy* for its corresponding Mission Space Object (MSO).

3.2.1.1 The Relationship Between a Model and its Federation Object -- a Very Simple Example.

Figure 3 shows a simplified object diagram for an MSO and its corresponding Federation Object. This diagram represents a very crude picture of the relationship between a model (in this case a Tank) and its Federation Object (the Tank FedObj). The Tank Model always keeps its own Tank FedObj object's state up to date, but the reflection of this state to other models is handled automatically by the OMF and Interest Manager (described later). Note that though many different models might make up a tank (i.e. Hull, Sensor, Brain, Weapon, Munition, etc.), only the model that has an independent existence on the battlefield and thus requires a "public face" (the Tank) has a corresponding proxy Federation Object. Federation Objects may also exhibit a composition hierarchy based on scalability and data transmission considerations (i.e. a separate "Kinematics" Federation Object for frequently updated information). The Tank Model also has expressed interest to the Interest Manager which manages a collection of other Federation Objects

(proxies) that meet the Tank's interest criteria. These Federation Objects are monitored by the tank. The Interest Manager dynamically updates these collections as the exercise progresses.

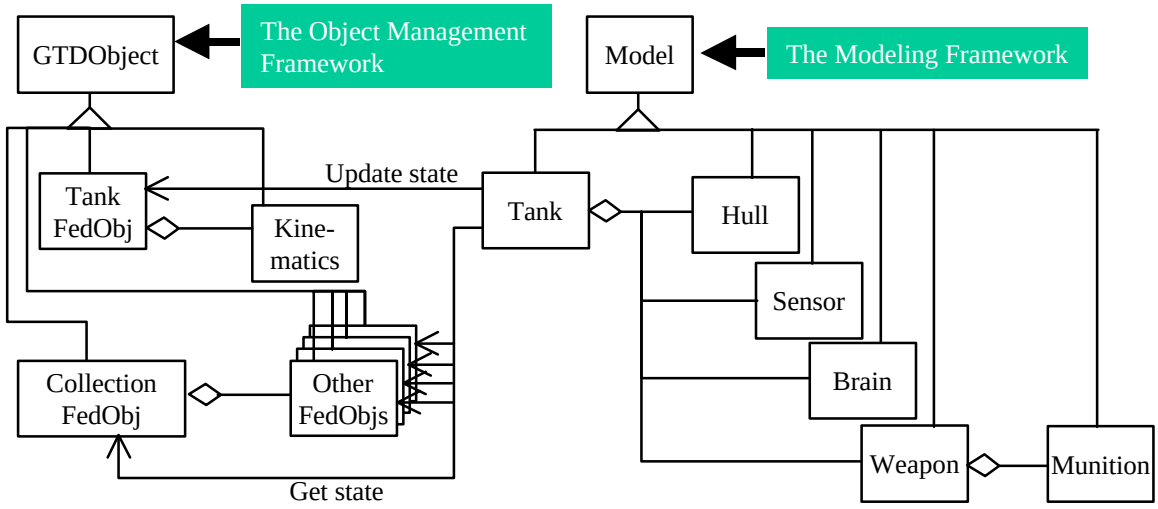

Figure 3. The Relationship Between a Model and Its Federation Object.

There are a few important concepts to understand about Models and Federation Objects:

- MSOs may be composed of other MSOs in a composition hierarchy, but only the MSO that has an independent existence on the battlefield (called the Master MSO) has an associated Federation Object.
- The Master MSO is responsible for keeping the corresponding Federation Object's state consistent and up-to-date.
- Federation Objects may also exhibit a composition hierarchy but this hierarchy may be completely different from the MSO composition hierarchy. Reasons for this difference in design are related to the different roles these objects play. MSOs are primarily modeling objects that need to be organized around modeling goals. Federation Objects are primarily state-containing objects that need to send the state to other MSOs and save the state to a database in the most efficient manner possible. Thus Federation Objects must be designed with efficient transport and storage in mind.
- The Federation Object is used throughout the entire exercise lifecycle as a means of exchanging information between *all* JSIMS applications, not simply those applications active during exercise execution time. Thus federation objects are called "end-to-end" objects because they exist from the beginning of the exercise lifecycle until the exercise is completely finished and all interesting information about it has been archived into the JMSRR.

3.2.2 HLA Run-Time Infrastructure

The HLA RTI is responsible for all run-time Federation Object communication and distribution. In general, the RTI performs the role of a generalized time-managed message-passing infrastructure. The RTI has six categories of interfaces and functionality:

- Federation Management. These functions manage the creation, management, and destruction of individual federation executions. In general, these functions cause the sending of control messages between applications.
- Declaration Management. These functions manage the publication and delivery of the appropriate data based on the FOM and the ability of a given application to understand a given object class. Declaration Management Services allow extensibility of the FOM without recompilation of every application.
- Object Management. These functions allow the transmission and receipt of the actual objects and interactions that make up the FOM.
- Time Management. These functions instruct the RTI how to order messages for delivery. In general, messages can be delivered in receive order, in time-stamped order with a guarantee that no message will arrive in the past, or in time-stamped order with no guarantee that a message will not arrive in the past thus necessitating the capability of "roll-back" on the part of the application. These three mechanisms correspond to the time management schemes known as *no time management*, *conservative time management*, and *optimistic time management*, respectively. The RTI supports

all three mechanisms with varying degrees of efficiency and performance.

- Ownership Management. These functions provide messages for managing the ownership of federation objects. An owner process is that process allowed to write the values for a given federation object. In particular, in the HLA, individual attributes of a given federation object can be owned by different applications. JSIMS will not use this particular feature of the HLA heavily. JSIMS will treat objects as the most atomic unit of ownership, but use complex composition hierarchies to achieve equivalent results.
- Data Distribution Management. These functions control the mechanisms for distributing the messages across the underlying network medium's available communications channels. In general, DDM controls what multicast groups messages are sent upon.

3.2.3 Common Data Infrastructure

The CDI is a value-added database managing persistent data through the exercise lifecycle. The CDI stores all data collected during an exercise and records cause and effect relationships among runtime data and between exercise generation data and runtime data. The CDI is also the primary means of storing and communicating information in pre-exercise and post-exercise phases. The CDI is not the JMSRR. A CDI exists for each exercise execution, while there is only one global JMSRR. During the exercise generation phase, objects and data are retrieved from the JMSRR and placed in the CDI to be used for a given exercise. Occasionally data from the CDI (such as pre-exercise-defined missions or post-exercise summary data) is stored back into the JMSRR after exercise execution. The CDI can be thought of as the scratch database for an exercise, storing initialization information, runtime collected data, and analytic results, while the JMSRR is the global repository for all exercises.

The HLA RTI provides a way to manage run-time simulation data. However, persistent data makes up the bulk of the data for any exercise: exercise generation / simulation initialization data with rich semantic content and high complexity but with low performance requirements and run time collected data of low complexity but large volume that requires high performance and places heavy demands on the available storage. Other forms of persistent data consist of legacy system data, meta-data, analyzed data, etc. The CDI fills the role of persistence engine for a given exercise. The CDI is:

- Distributed. The CDI is a federated multidatabase. It exists on multiple computers, but to the application developer it acts as if it were a single monolithic database. This greatly simplifies data collection and analysis, because it allows distributed data collection while permitting analysis software to act as if the database were all in one location. Certain types of queries (those resulting in distributed joins) have poor performance in a federated multidatabase. Users are warned when attempting to query the database in a manner that will result in a distributed join.
- Active**.** The CDI accepts and responds to queries posted into the future, greatly enhancing exercise management and analysis capabilities. The mechanism for implementing active queries is through the use of agents running on data collector applications. These agents wait until they detect a particular event, then evaluate a condition for validity. Given a valid condition they then perform some action. This is called the Event-Condition-Action paradigm and it will be implemented in the CDI based upon the JSIMS sequencing of requirements.
- Spatio-Temporal. The CDI explicitly indexes data based on geography to support the most common queries needed to assemble an AAR. Temporal indexing permits efficient storage and retrieval of end-to-end objects.
- Object-Oriented. The CDI presents its data to an application as objects. The CDI is designed to be an object-relational database, including the strengths of both object and relational DBMSs.

Access to the CDI is done using a number of special mechanisms tailored to the needs of the user. Primarily, there are four users of the CDI, each with a different requirement on the database.

Exercise Generation applications require access to the database as if it were an object-oriented database with a high degree of navigation capability built in augmenting the standard query capability. For these applications, the CDI software in the JOS translates between the underlying relational nature of the Oracle database and presents to the application a federation object of the type requested. Similarly, the CDI software translates a given application's modified federation object into the underlying SQL statement that is used to store the object's state into the database. The schema for storage of an object may not bear a direct resemblance to the in-memory C++-like structure that an object has when activated. The schema may be optimized for performance both at exercise generation time as well as during collection at run-time. SQL is used for accessing the database in this case.

The second type of CDI user is the JSIMS simulation applications containing the mission space objects that need to fetch their initial state from the corresponding federation object at initialization time. These applications need to be able to get their initial state and then instantiate the run-time portion of their federation object in one seamless operation. Essentially, at initialization time, mission space objects must use both the initialization interface and the run-time evolution interface of the end-to-end

federation object at the same time. SQL is used as the underlying mechanism for accessing the database in this case.

The third CDI user is the data collectors, which gather run-time federation object state evolution information and store it into the CDI for later analysis. An efficient and tailored schema greatly helps realize maximum performance. Because of the high-volume nature of the collected data, SQL must usually be bypassed in favor of custom mechanisms for storing data into the database during run time. During such database loads, all database transaction functionality is bypassed, although some creation of indices may continue.

The fourth user of the CDI is the analysis and AAR applications that query the database for information related to illustrating what transpired during previous portions of the exercise. These applications may access the data in a number of different fashions. First, using the temporal interface to a given end-to-end object, the application can view a given object as a unified temporal whole stretching from its initial state to its final state and including the documented reasons for state evolution. Secondly, an analysis application may choose to perform direct SQL queries against the database and interpret the resulting table in a fashion consistent with the application's understanding of the database schema. In this way, numerous COTS tools such as PowerBuilder or Microsoft Excel can be linked to the database using commercial standards such as ODBC or JDBC. The limitation of this mechanism is that the application must reconstruct the federation object on its own without help from the CDI software. The benefit of this mechanism is the ability to leverage commercial products for analyzing and reviewing an exercise.

3.2.4 Local Cache of the Ground Truth Database and the Object Management Framework.

The JOS provides an in-memory database of federation objects for each application to use. This database contains all the federation objects corresponding to local Mission Space Objects as well as all federation objects that are proxies for remote mission space objects. Given that JSIMS may have an enormous number (10^5 to 10^6) of objects active at any one time, it is necessary to limit each process's database to only those objects that the local MSOs are interested in. Thus one can think of the union of all the federation objects distributed among all simulations in a given exercise as the "Ground Truth Database," while each individual process only has a local cache of the Ground Truth Database (GTD). The GTD exists only in memory, not in a persistent store. Thus it is distinct from the CDI which is the persistent equivalent of the GTD.

The GTD is created from the Object Management Framework illustrated in Figure 4 below. The Object Management Framework (OMF) is the collection of base classes from which Federation Objects are derived. Each Federation Object inherits from the GTDObject base class. Event Handlers are the equivalent to GTDObjects for Interactions (Fire, Detonate, Signal, etc.) Event Handlers allow MSOs to register for asynchronous notification of the delivery of an interaction.

Views are objects that implement the "Observer" pattern with respect to GTDObjects. When a GTDObject is altered, any Views that are attached to the object are activated. Views do not run synchronously with the object that has been changed; they are scheduled to run in the future. This scheduling prevents a view from being activated when for example *location.x* is changed and then again immediately after that when *location.y* is changed. When a scheduled View does run, it runs only once per observed object no matter how many times it has been activated. There are a three different types of views defined in the OMF: the Reflector View used to schedule a GTDObject for reflection out through the RTI, a Collector View used to schedule a GTDObject for storage into the CDI, and a Trigger View used to provide asynchronous notification to an MSO that a particular GTDObject has changed. Each simulation application maintains a single local cache of the GTDObjects of interest to all the MSOs present in that simulation. Collections of GTDObjects are themselves GTDObjects with all the requisite properties and behaviors (including the ability to have Views attached to them). The Interest Manager maintains a collection of GTDObjects for each Interest Expression it has that is active. The requesting MSO or Application is given a reference to the particular collection that corresponds to its interests. An MSO can attach a Trigger View to its collection to allow asynchronous notification of any changes to the collection, including changes in the collection's cardinality.

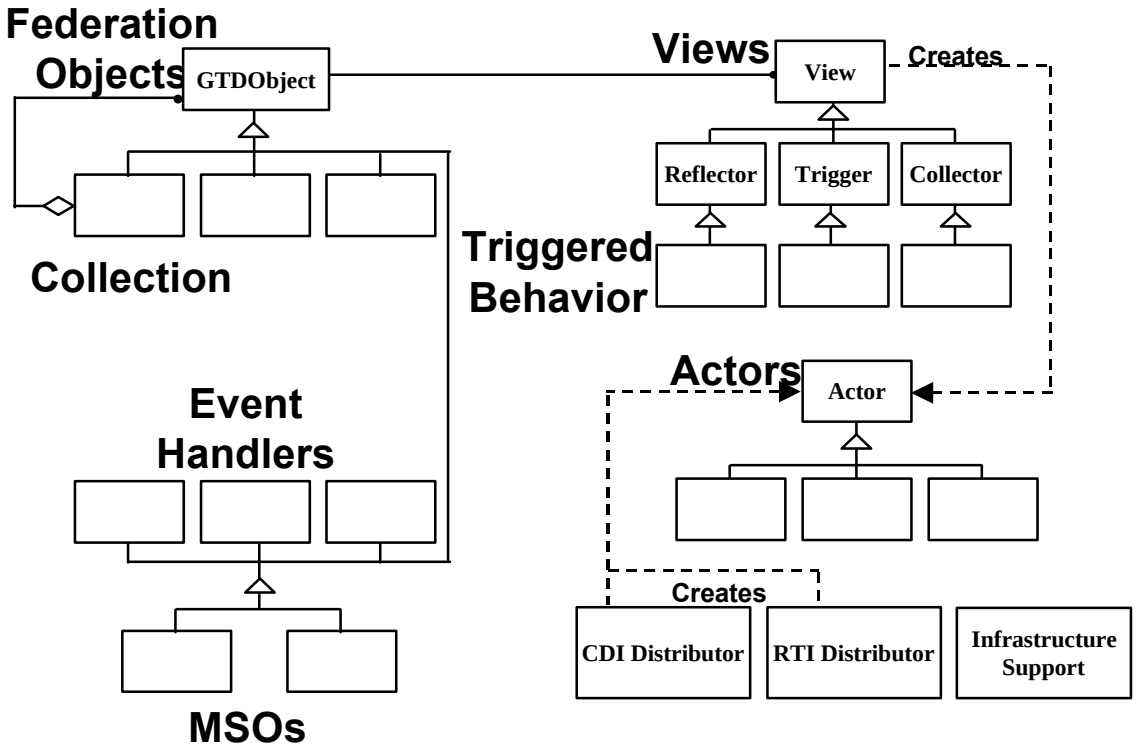

Figure 4. The Object Management Framework implementing the Ground Truth Database

Actors perform all the workhorse functionality in the OMF. They interpret and translate to and from the RTI and CDI, factory objects, and update the GTD when new information arrives. Actors contain state and behavior to update GTDObjects and communicate interactions. Special Name-Value Pair Actors will facilitate rapid RTI integration and improve performance. Distributors are examples of the façade pattern that shield the application from difficult-to-use interfaces, such as for the RTI and the CDI.

3.2.5 Interest Manager

The Interest Manager is the scalability engine of the JOS. It has the responsibility of making the decisions required to route outgoing data to the appropriate places as well as ensuring that its particular application receives all the information it requires to function properly. The Interest Manager also sorts incoming data into convenient collections for the use of the application. All this automation, however, is not without its cost. MSO developers must be very cautious in how they use the Interest Manager as significant performance penalties can be the result of incorrect usage. The Interest Manager is responsible for four basic tasks:

- Interest Expression. All of the MSOs running in a particular process constitute a JSIMS application. In general, the application understands what type of data it is interested in. Using methods provided by the Interest Manager, the application can formulate an Interest Expression (IE) that precisely describes this interest. The Interest Manage is then responsible for communicating this IE to the RTI in such a way that the RTI will deliver data to the GTD that fulfills this interest. The application may create many IEs or just a single IE. There can be multiple IEs for each MSO in the application, a single IE for all MSOs in an application, or anything in between.
- Routing of Outgoing Data. When a locally-owned Federation Object's state is updated and it has a Reflector View attached, that object's state will be scheduled to be transmitted to any interested parties via the RTI. When the Reflector View executes, it asks the Interest Manager for information about what logical destination the data should be sent to. Since the data is being sent via the RTI and the RTI uses an abstraction known as "Routing Spaces" for the underlying network multicast groups, this functionality consists of basically deciding, given an object that needs to be transmitted, what part of Routing Space should the object be sent to (i.e. onto which multicast group should it be transmitted). The Interest Manager has a method that performs this function based on a particular multicast strategy that has been selected for a given exercise.

- Filtering and Sorting of Incoming Data. When the RTI delivers an incoming piece of data, a number of tasks are performed. First the data is used to either create an interaction, or it is used to create a new or update an existing object in the Ground Truth Database. In either case, the Interest Manager is used to help decide where else to route the incoming information. The Interest Manager has a list of all active Interest Expressions for a given application. For objects in the GTD, the Interest Manager maintains a collection of GTDObjects for each Interest Expression. Each collection contains all and only those objects that fulfill that IE. Thus as new data is received the Interest Manager must update all relevant collections so that they are consistent with the new information. In the case of Interactions, the Interest Manager uses its list of IEs to identify the appropriate Event Handler(s) that must be called on receipt of that particular type of interaction. The Interest Manager performs these sorting and filtering functions on incoming data to relieve the MSO designers of having to worry about culling through the GTD each time the MSO is active trying to find only those GTDObjects that are relevant to that particular MSO. Such a searching/sorting process can be extremely time-consuming and cause a lot of wasted processor effort. In effect the Interest Manager performs an efficient insertion sort on incoming data to make sure every collection is kept up to date. Unfortunately, the amount of time the Interest Manager spends keeping the collections coherent is directly related to the number of Interest Expressions active in a given application at a given moment. Thus it becomes very important not to create a large number of overlapping Interest Expressions that cause the Interest Manager to have to perform a large amount of work on the receipt of each new piece of information. Nevertheless, properly used, such a sorting scheme can be extremely efficient in ordering data such that MSOs can know exactly which Federation Objects are relevant to them and which are not without searching the GTD.
- Managing the Interaction of Predictive Contracts and Interest Expressions. "Predictive Contracts" is the name given to the mechanism of MSOs signing a "contract" for their behavior in the future. As long as they do not "break the contract," they will never cause information to be transmitted via the RTI. In general, Predictive Contracts are a form of replicated computing, based on the fact that it might be less expensive to perform redundant calculations on multiple processors rather than send information across a network. Because with current technology it is very expensive to write to and then read from a network, replicated computing can be an efficient implementation strategy for message-passing simulations such as JSIMS. The most common form of Predictive Contracts is the Remote Vehicle Approximation or "Dead Reckoning" used in the DIS Standard. With Dead Reckoning, an entity promises not to deviate from a linear (or quadratic) path above a given threshold without informing any listeners of such a deviation. This type of Predictive Contract can reduce network traffic by an order of magnitude. More sophisticated Predictive Contracts are theorized to be able to reduce network traffic by another order of magnitude with little increase in computational cost. Unfortunately, the use of Predictive Contracts can interfere with the use of Interest Expressions if they are not handled properly. For example, an entity publishes a predictive contract consisting of its route of travel. This route begins outside of any interest expression, travels through areas that other MSOs have expressed interest in, then exits all relevant interest regions. In this case the entity will publish the contract and its current position and then no more information until it either breaks the contract or finishes the route specified in the contract. Thus without the intervention of the Interest Manager, any MSOs that have expressed interest in the regions through which this entity travels will not be informed of its relevance to their interests. The Interest Manager, therefore, is responsible for analyzing any Predictive Contracts that are communicated, and predicting when an extrapolation based on a contract will intersect with any active interest regions. The Interest Manager then sets a timer to expire when this intersection occurs and places the relevant entity's federation object in the appropriate collection based on its current position obtained from the contract.

JSIMS will extensively use Predictive Contracts as a mechanism for limiting network transmissions. However, these contracts will almost always be based on an MSO's position as a function of time.

3.2.6 JOS Interfaces

There are five categories of interfaces to the JOS:

- The CDI interface. This is the interface that provides direct access to the CDI database. Two mechanisms are used: direct SQL access, and OO wrapped access that allows a method to be invoked and a Federation Object to be returned.
- The Object Management Interface. This interface has methods to allow the creation and destruction of Federation Objects, the creation and management of collections and iterators, and the overall management of the JOS.

- The Object Interface. This is the set of methods that allow access to each Federation Object and Interaction
- Interest Manager Interface. This interface allows the formulation of Interest Expressions and the manipulation of the collections associated with each Interest Expression.
- The RTI Distributor Interface. This is the façade that allows access to the RTI's time management, federation management, and ownership management functionality.

4. INTEGRATING OTHER SYSTEMS INTO A JSIMS COMPOSITION

To meet JSIMS's requirements it is necessary to make use of a number of systems that are not under the control of the JSIMS program. In particular, JSIMS requires that the training audience interact with the JSIMS simulation using their go-to-war C4I systems. Also, it will be necessary from time to time to federate JSIMS with other simulations or simulators. The Translation Services are designed to translate between JSIMS native "language," the Federation Objects, and the language spoken by these other systems.

4.1 Other Simulations: the Gateway Approach

Gateways provide the linkage between JSIMS and other simulations or simulators. They utilize the HLA and the RTI to federate JSIMS with these other systems. The JSIMS internal FOM is not changed, but is treated as its SOM for the purposes of federation. This approach makes JSIMS *externally* HLA-compliant as well as *internally* HLA-compliant. An illustration of the Gateway Approach is shown in Figure 5 below. With this approach, all components use the RTI to communicate, guaranteeing HLA-compliance. There are two FOMs: one internal to JSIMS, one between JSIMS and the external system. The peculiarities of the external system are encapsulated in one place, thus changes in the external system require only one piece of software to be modified. This solution can be replicated *ad infinitum*.

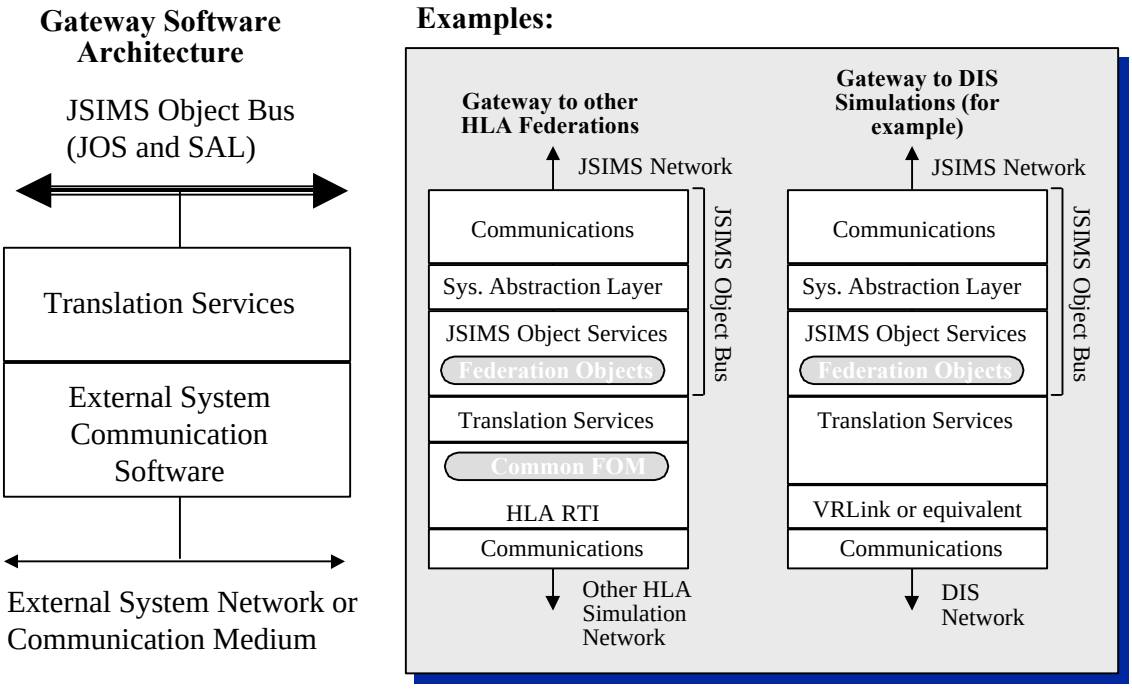

Figure 5. Software Architecture of External System Gateways.

4.2 C4I Systems: the Surrogate Approach

The Modular Reconfigurable C4I Interface (MRCI) Program, sponsored by the Defense Modeling and Simulation Office (DMSO), provides a number of lessons learned for incorporating operational C4I systems into JSIMS. MRCI has developed a suite of translators that translate between numerous operational message formats and the Command and Control Simulation Interchange Language (CCSIL), a machine-interpretable language used to communicate command and control information.

All training audience interfaces to JSIMS will be via operational C4I systems. Messages are translated between operational formats and CCSIL and affected by simulations of the tactical communication systems used. C2 surrogates simulate physical C4I systems in the simulated world. Simulated C4I systems and decision makers are fully integrated into the simulated world .

4.3 Lifecycle Framework and Common Services

All Lifecycle Applications are built on a common Lifecycle Framework. This framework provides reusable software components such as map displays, graphics primitives, mechanisms for interfacing with the JOS, performing time management, etc.

4.4 Lifecycle Applications

Lifecycle applications automate exercise construction, execution and review with reduced overhead. There are a number of lifecycle applications:

- Exercise Planning Tool: this tool aids the exercise designer in planning the exercise objectives.
- Scenario Analysis Tool & Infrastructure Analysis Tool: This tool suite automates scenario investigation and resource management by providing a "simulation of the simulation" and "simulation of the system" capability.
- Exercise Generation Tools: a suite of tools that automate creation of exercise initial conditions
- Data Collection Management Tool: automates data collection planning and the management of the data collectors during run-time
- Data Collectors: these applications collect the run-time simulation data for later analysis and review.
- Exercise Management Tools: these tools perform the functions of simulation control, technical control, exercise observation
- Exercise Analysis and Review Tools: these tools automate preparation and presentation of AARs, and aid understanding the exercise technical results.
- Archiving Tool: this tool automates the archival of information into JMSRR

Each of these tools incorporates key automation technologies. The Exercise Lifecycle Tool Suite represents a single integrated system based on a common framework that provides a uniform method of technical control. The tools all have a common look and feel based on an advanced user interface. The tools are fully integrated with operational C4I systems. They also have the ability to view and extract data from the JMSRR. Exercise planning tools contain advanced automated decision-making software that aids the user in constructing an exercise. This automation inherent in the Exercise Lifecycle tool suite is critical to reducing the amount of labor currently required to design, build, run, analyze and review an exercise.

Lifecycle Applications can have their GUI run on a separate platform from their main executable. This allows more flexibility in being able to design an exercise since it allows the GUI for the exercise generation tools to be located at a user site, while the main application is located near the data that it needs to use. This approach greatly improves application performance by limiting the amount of information that must be transmitted over a Wide Area Network, while simultaneously allowing the users to run the application from anywhere in the world. The Lifecycle Common Services allow the applications to tie into DII COE common modules when this is appropriate so that these modules can be reused in JSIMS.

5. MODELS AND THE MODELING FRAMEWORK.

Models are the mission space objects based on the Joint Conceptual Model of the Mission Space (JCMMS). They represent:

- Forces (service specific and joint, unit and platform)
- Environment
- Intelligence, Surveillance, and Reconnaissance
- Command and Control decision makers and behavior

All models connect to the JSIMS infrastructure through the modeling framework. Model composition occurs at the modeling framework interface. Models can be as complicated or as simple as desired, and can support multiple resolutions depending on the needs of the particular exercise composition.

5.1 Modeling Framework

The Modeling Framework enables model creation in a uniform, consistent, composable fashion. The object-oriented framework gives flexibility, extensibility, and aids in creating interoperable models. Having a single uniform Modeling Framework ensures that the MSO developers build their MSOs in a consistent fashion.

The Modeling Framework is a set of interacting base classes that span the problem space of model development. Each base class defines a number of interfaces and the relationships between the interfaces. Model developers subclass from these base classes, override and implement the interfaces and the model's methods are then called at the right time.

5.2 Mission Space Objects

Mission space objects populate the simulation battlespace and provide life to the training exercise. Mission space objects are bound and described by several architectural principles:

- Mission space objects inherit from base classes defined in the modeling framework
- Mission space objects have a close relationship with their corresponding Federation Object

- Mission space objects can be executed in logical sets

Mission space objects inherit from the modeling framework. The framework provides an overall architecture for how models are executed and what models look like (the framework is a "template" that provides a starting point for developers). The model developer inherits primitive capabilities such as scheduling primitives and message-passing capabilities. However, other functional capabilities are provided via direct calls to the JOS and various toolkits and libraries.

Mission space objects have a close relationship with the FOM. Simulated things (entities or aggregates) in the mission space are composed of some set of components (where components are functional capabilities represented in the design as abstract classes). Thus, a "tank" is composed of a "hull", "sensors", etc. A given run-time instance of a simulated thing is actually executing a set of models, where models are represented in the design as concrete classes derived from the components. Using this terminology, a "simulated thing" is an object that has a distinct physical presence on the simulated battlefield and thus requires a “public face” for interaction with other simulated things. This public face is the thing’s associated Federation Object.

Thus, each Federation Object is defined in conjunction with its associated Mission Space Object and is implemented to match the attributes that can be exported by simulated things operating at different resolutions.

Mission Space Objects can be executed in logical sets. A logical set of mission space objects must be formed so that a given exercise execution makes sense. Exercise generation tools must limit the exercise to a logical combination of models. There are three limitations that bind the logical model sets supported by JSIMS.

- A limited number of models can be developed (since not all possible resolutions of all possible things can be implemented, only a subset).
- Models in a given exercise must be able to communicate together both via Federated Objects they both understand and via direct method invocation. As an example of the former, a very high resolution thermal sensor will not get sufficient data from a constructive "division", whose Federation Object may contain center of mass data, not temperature profiles of engine exhaust.
- Models that are otherwise logically interoperable may not be able to operate together given constraints on available resources or exercise execution parameters. An example is an attempt to run with too many high resolution entities (or with too many medium resolution entities to handle a given faster-than-real-time execution rate).

Models can be directly controlled by the Senior Controller Workstation. That is, each model can receive specific messages from and respond to the Senior Controller Workstation. This capability will allow the Senior Controller Workstation to be able to directly change a model’s internal parameters. This capability must be implemented using specially-defined Federation Object Interactions, so that such changes can be recorded by the Data Collectors in the CDI for later analysis and review.

5.2.1 Submodel to Submodel Communication

Communication between submodels will be by both direct method invocation and by the passing of messages depending on the requirements of the time management technique that is designed and implemented for JSIMS. Certain communications between submodels can be synchronous (i.e. not time-managed). These communications can be implemented as direct method invocations. Other communications between submodels that must be scheduled (and thus time-managed) must be implemented via message-passing rather than direct method invocation. The distinction between these two techniques, both of which shall be supported by the Modeling Framework, will be documented in the SSDD.

6. JSIMS MODELING AND SIMULATION RESOURCE REPOSITORY (JMSRR)

The JMSRR essentially consists of four environments that are supported by a distributed infrastructure and information security.

- User Environment - The end-user environment in which users interact directly with the repository to perform basic repository operations or to administer the repository.
- Application Environment - The technical environment that supports client applications for domain-specific functionality (e.g., JSIMS Lifecycle Applications for building, analyzing, and recording an exercise).
- Common Services - “Central” common services that support the application environment with remote applications or functionality and also support internal operations such as cataloging, indexing, and routing of information flow.
- Libraries - JMSRR Libraries contain all of the repository content along with the metadata, contexts, and tools that support domain-specific use of the content.

From a development perspective, this arrangement of environments also defines three areas of interest: Producers (those who provide the library content), Consumers (those who use the content, both users and domain applications), and Infrastructure (those

interested in the technology on which the repository operates).

The previously mentioned environments or logical system "layers" are shown in Figure 31 mapped to the Hierarchical Virtual Repository paradigm. An additional infrastructure consideration specifically addressed by the JMSRR are the "External Interfaces". Because the JMSRR is part of a larger information space, the MSRR, the JMSRR will maintain at least one external interface to support information flow between the MSRR and the JMSRR so that users of either system can seamlessly access at least a subset of the other. Over time, there might be other external systems to which the JMSRR will provide an interface (not all will necessarily provide two-way information flow). Realms are logical security domains that can be synonymous with physical sites to support dynamic security profiles associated with users, geographical location, facilities, specific exercises (time, purpose, "need to know"), etc.

7. A UNIFIED APPROACH TO MANAGEMENT OF PERSISTENT DATA THROUGHOUT THE EXERCISE LIFE CYCLE

In past simulation systems, the management of exercise data was treated in a stovepipe fashion, with many databases, and database management systems. This method impeded interoperability, flexibility, and extensibility of these simulation systems. In particular, there were many mismatches between different types of data, between different database systems, and there were important inconsistencies between data used to create an exercise and the data that was captured during an exercise, leading to numerous problems interpreting exercise results. The JSIMS architecture provides for a uniform mechanism for managing data throughout the exercise lifecycle (illustrated in Figure 10). The JSIMS approach provides for uniform data management throughout the exercise lifecycle using the CDI as a single (though distributed) common database and an end-to-end object model (described in the next section) for defining the schemas of objects that go into the database.

The benefits of a unified approach to lifecycle management of persistent data are numerous. First, with a single system, data relationships are more easily maintained, allowing for much easier interpretation of the data during analysis. Second, there is no "impedance mismatch" between pre-exercise and run-time data, causing potentially erroneous interpretation of information and forcing the user to create numerous "translators" that translate between the different data formats. Third, cause and effect traceability is significantly enhanced because information collected during run-time can be more easily related to information that existed in the Exercise Generation phase of an exercise. Finally, it is much easier to archive pre-exercise, run-time, and post-exercise data into the JMSRR because there is a single object-oriented data format to the exercise data that is compatible with the JMSRR architecture.

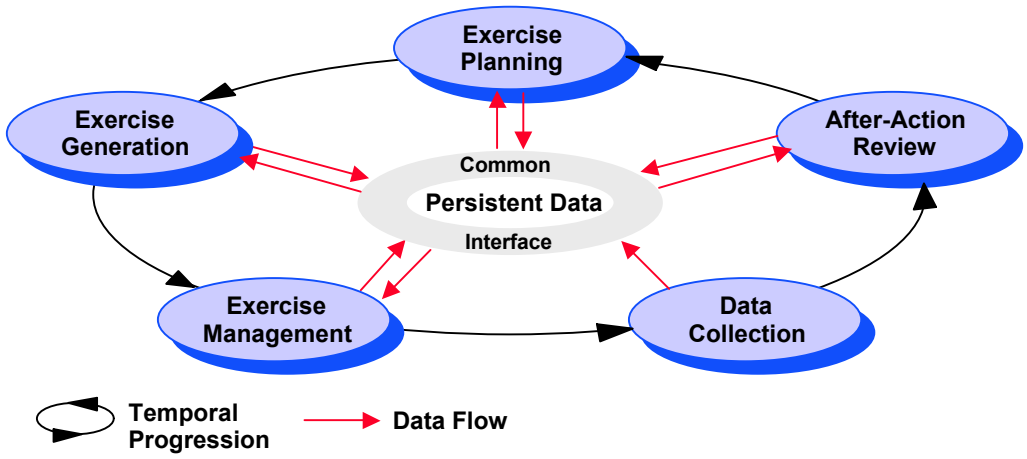

Figure 6 A Uniform Approach to the Management of Exercise Data

8. BENEFITS OF THE JSIMS ARCHITECTURE

Our approach to JSIMS emphasizes the use of advanced software technology to create a more flexible, extensible, reliable, maintainable, and interoperable JSIMS system. OO Frameworks give the system flexibility, extensibility, and reusability by providing structure for solutions to problems, while allowing modelers to subclass and extend the Framework to create specific functionality. They also enable the construction of a composable system, with its concomitant benefits of lowered overhead and ease of exercise construction, execution, and review. End-to-end object

management yields interoperability throughout the entire exercise life cycle by defining a single common language, the Federation Objects, that all applications understand. The Life Cycle Applications provide users with automated tools to quickly and easily compose exercises to meet their requirements while lowering exercise overhead.

As this paper points out, there is a considerable amount of work and effort to develop a set of flexible software that allows the simulation developer to follow the lead of the CBT programmer. Figure 7 represents the idealistic view of JSIMS. The simulation developer needs to focus only on the value added portions of the system, not reimplementing the infrastructure. By doing this, the simulation return on investment will be maximized.

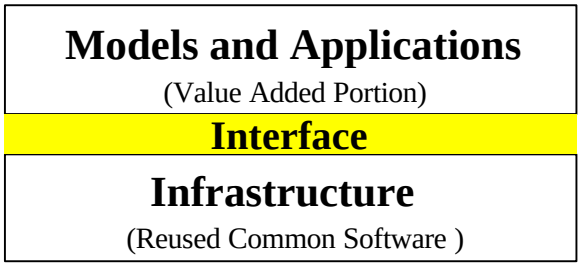

Figure 7 The Idealized Model of the JSIMS Software Architecture

9. MORE INFORMATION

Considerable more detail is available concerning the JSIMS Software Architecture, as well as the program as whole, is available from the JSIMS home page located at http://www.jsims.mil. This paper is extracted from the JSIMS Architecture Definition Document Version 1.0, which is also on the web.

10. ABOUT THE AUTHORS

Dr. Edward T. Powell is a senior scientist with Science Applications International Corporation. He received his Ph.D. in Astrophysics from Princeton University. After four years at the Lawrence Livermore National Laboratory's Conflict Simulation Laboratory where he worked with the Joint Conflict Model, he moved to SAIC and became the chief simulation infrastructure architect for the largest operational warfighter DIS exercise ever created, the 10,000+-entity live/virtual/constructive JPSD-96 exercise. He then became a lead architect on the ARPA-funded Synthetic Theater of War Program, where he oversaw the creation of the STOW Simulation Infrastructure and the STOW Data Collection System and CDI. He is now the principle architect and Core Infrastructure AIT Leader for the Joint Simulation System Integration & Demonstration contract.

Dr. David R. Pratt is the first Technical Director of the Joint Simulation System (JSIMS) Joint Project Office in Orlando, Florida. He holds this position concurrently with an appointment as an Associate Professor at the Department of Computer Science, Naval Postgraduate School in Monterey, California. As the Technical Director, his responsibilities include the overall technical direction of the program and the investigation of applicable technologies relevant to large scale constructive simulations. His academic interests include real-time 3D computer graphics, software architectures, and distributed computing. Dr. Pratt lead the development of the NPSNET system, one of the first publicly available DIS based virtual environments. Prior to joining the faculty at NPS, Dr. Pratt was a Data Processing Officer in the United States Marine Corps. His has an extensive publication record with over 40 published articles covering a wide range of computer topics. Dr. Pratt holds a Ph.D. and M.S. in Computer Science from NPS and a BSE in Electrical Engineering from Duke University.